Finding Lane Lines on the Road

Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file. But feel free to use some other method and submit a pdf if you prefer.

The goals / steps of this project are the following:

- Make a pipeline that finds lane lines on the road
- Reflect on your work in a written report

Reflection

1. Describe your pipeline. As part of the description, explain how you modified the draw_lines() function.

My pipeline consisted of the following steps.

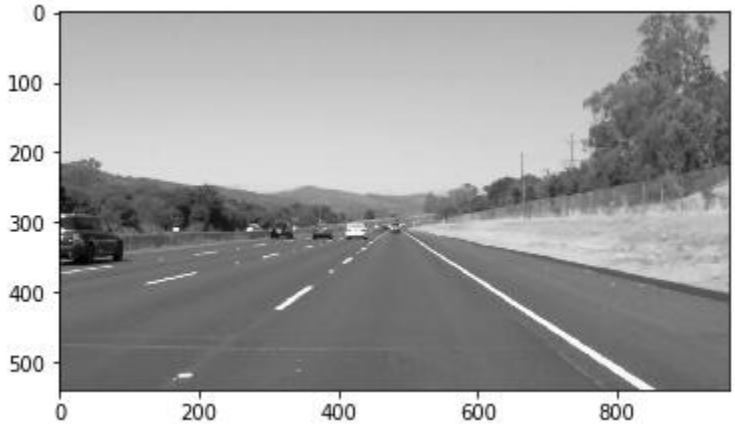
Step 1: Read the image

image = mpimg.imread('test_images/solidWhiteRight.jpg')

```
This image is: <class 'numpy.ndarray'> with dimensions: (540, 960, 3)
Out[2]: <matplotlib.image.AxesImage at 0x7f2ddbb7a400>
```

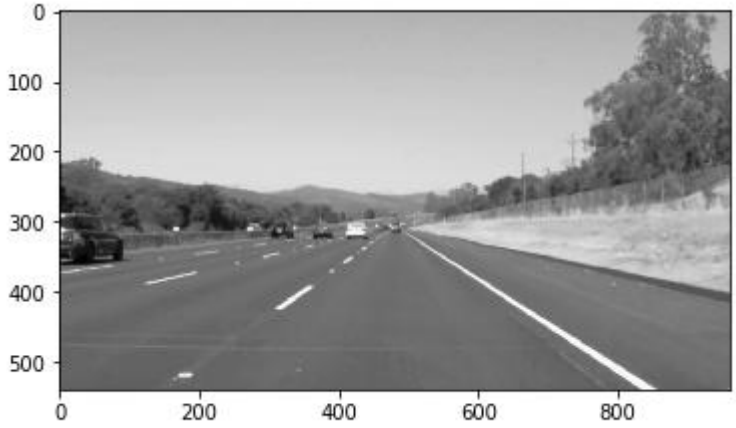
Step 2: Apply the Grayscale transform

```
def grayscale(img):
    return cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
```



Step 3: Apply a Gaussian blur with a 3 x 3 kernel to help remove high frequency noise

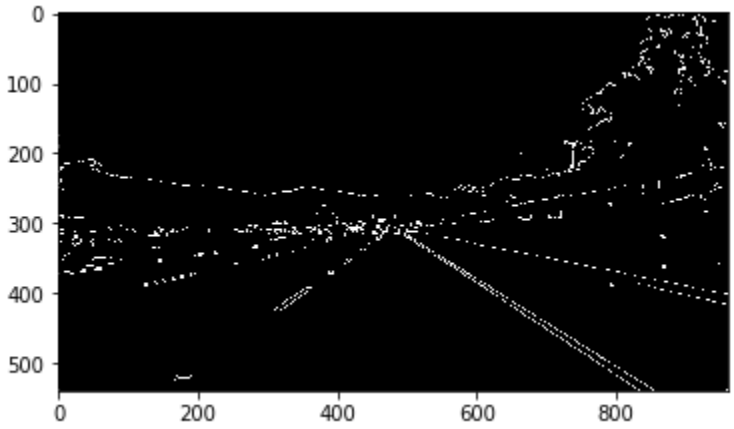
```
def gaussian_blur(img, kernel_size):
    return cv2.GaussianBlur(img, (kernel_size, kernel_size), 0)
```



Step 4: Canny transform using threshold based on the median

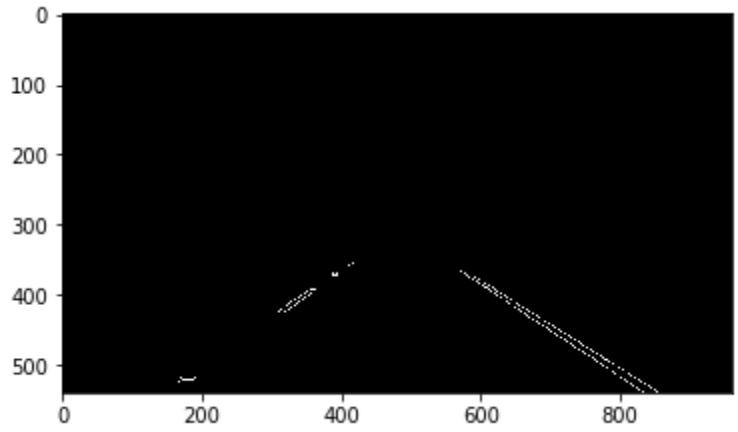
```
def canny(img, low_threshold, high_threshold):
```

```
    return cv2.Canny(img, low_threshold, high_threshold)
```



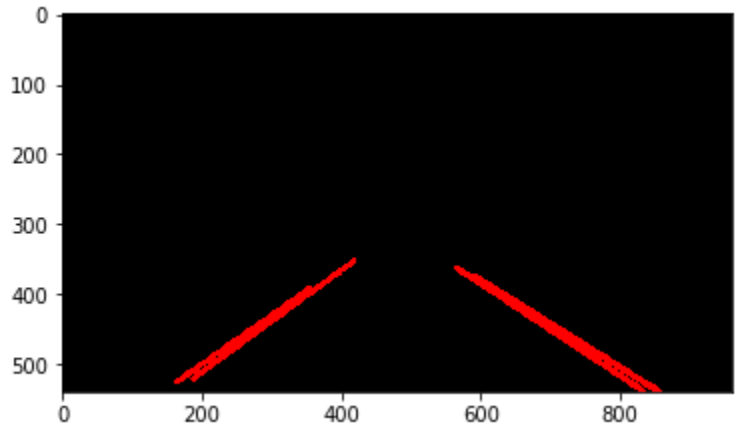
Step 5: Region of Interest masking

```
def region_of_interest(img, vertices):

    mask = np.zeros_like(img)

    if len(img.shape) > 2:

        channel_count = img.shape[2]  # i.e. 3 or 4 depending on your image

        ignore_mask_color = (255,) * channel_count

    else:

        ignore_mask_color = (255,)

    cv2.fillPoly(mask, vertices, ignore_mask_color)

    masked_image = cv2.bitwise_and(img, mask)

    return masked_image
```



Step 6: Hough Transformation

```
def hough_lines(img, rho, theta, threshold, min_line_len, max_line_gap):

    lines = cv2.HoughLinesP(img, rho, theta, threshold,

                            np.array([]), minLineLength=min_line_len, maxLineGap=max_line_gap)

    line_img = np.zeros(img.shape, dtype=np.uint8)

    draw_lines(line_img, lines)

    return line_img
```



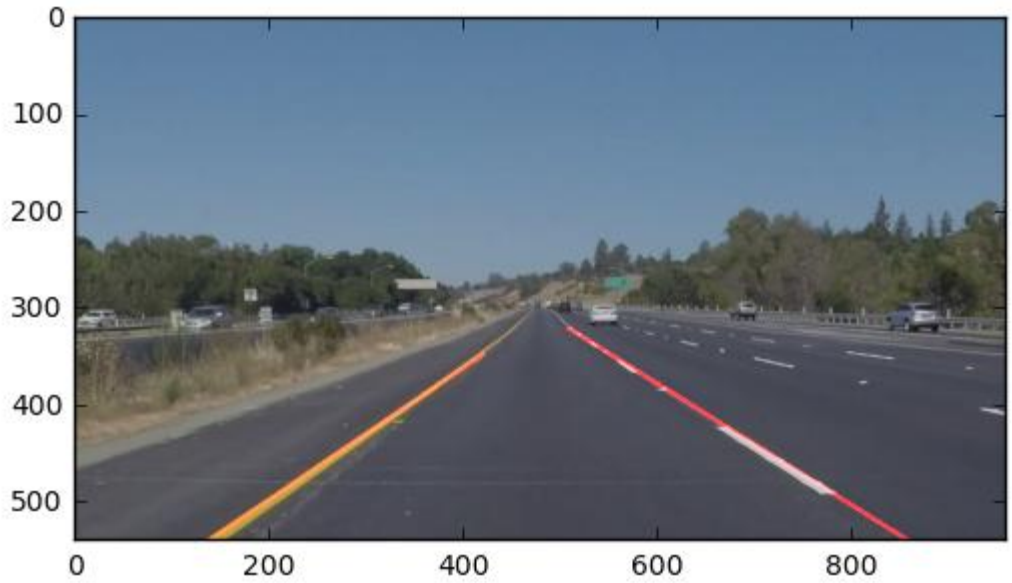
Step 7: Merging original image with lines

```
def weighted_img(img, initial_img, alpha=0.8, beta=1., gamma=0.):
```

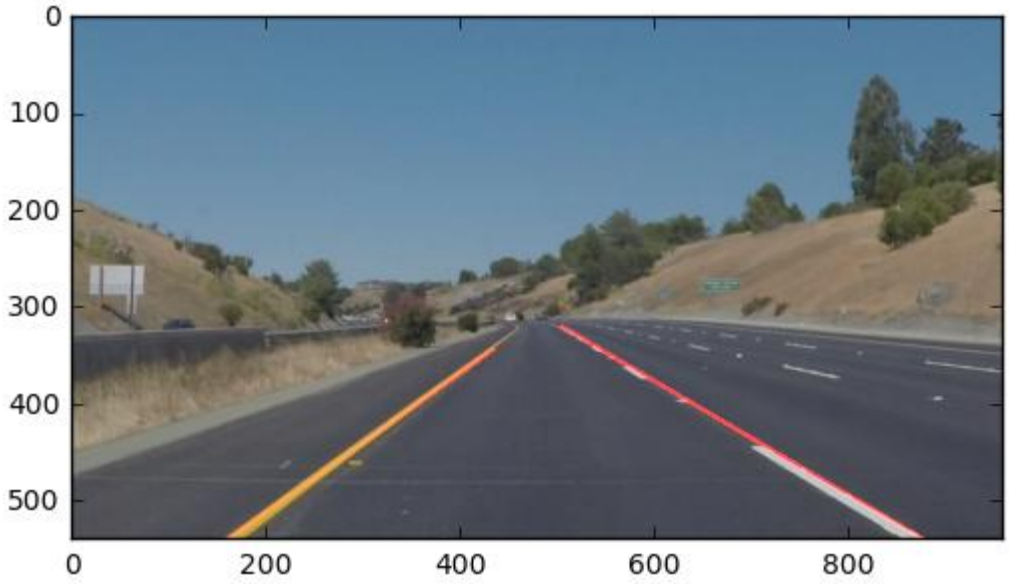
return cv2.addWeighted(initial_img, alpha, img, beta, gamma)

Step 8: Improve the draw_lines() function to average/extrapolate the line segments you detect to full extent of the lane.

Processed solidYellowLeft.jpg

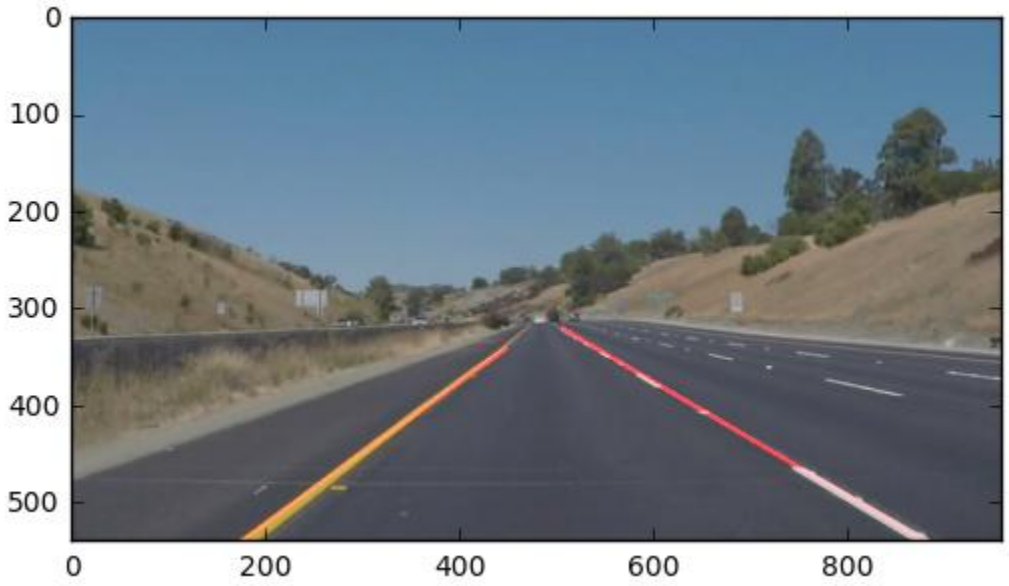


Processed solidYellowCurve2.jpg



Processed solidWhiteRight.jpg

Processed whiteCarLaneSwitch.jpg



Processed solidWhiteCurve.jpg

Step: Create loop to cycle through all the images

2. Identify potential shortcomings with your current pipeline.

The averaging of the slope may not allow the pipeline to work optimally on road lines that have strong curves.

3. Suggest possible improvements to your pipeline.

More accurate tuning of the Hough Transform parameters as well as the Canny edge parameters could help improve the outcomes.

Apply distortion correction techniques to the problem.